Interconnect Pipelining in a Throughput-Intensive FPGA Architecture

Amit Singh, Arindam Mukherjee, Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106
{asingh,arindam,mms}@guitar.ece.ucsb.edu

Abstract

Wave-steering is a new design methodology that realizes high throughput circuits by embedding layout friendly synthesized structures in silicon. In the wave-steering design methodology, circuits inherently utilize latches. Inside the synthesized structures they are used for signal skewing, and on the interconnects to guarantee the correct arrival times at the inputs. Recently, we proposed a novel high-throughput FPGA architecture based on the wave-steering design principle to handle throughput-intensive applications. Previously our work was focussed mainly on the Logic Block (LB) design. In this paper we discuss a pipelined interconnect scheme to support the strict timing requirements that is necessitated by the wave-steered design style. We characterize designs that best fit the new architecture and show that as technology scales down towards deep submicron (DSM), this FPGA fabric shows an increasing throughput performance.

1 Introduction

Since 1985, FPGAs have become increasingly popular for their ability to be a low cost solution in a variety of design applications. The advent of DSM technologies has given rise to million gate FPGAs, making them increasingly versatile. In addition, FPGAs, unlike custom logic, offer design flexibility by their ability to reconfigure. However, most commercial FPGAs cannot handle applications that require very high throughput. These throughput-intensive applications mostly occur in the real-time Digital Signal Processing domain. This inability to handle throughput-intensive applications is caused by the fact that most FPGAs have a general purpose architectural nature which forces them to be much (as much as ten times) slower than custom logic.

We proposed a throughput-intensive FPGA architecture fabric in [19], which targeted regular circuits. This fabric used a wave-steering [15][16] approach to implement circuits in pass transistor logic (PTL) mapped decision trees achieving throughputs which were an order of magnitude higher than those achieved in present day FPGAs. Wave-steering, unlike other synthesis techniques, naturally integrates logic and physical synthesis steps and takes the circuit clock period (throughput) as one of the input specifications. Wave-steering can pipeline a Logic Block (LB) to the granularity of one level in the tree, which is controlled by a single variable(input signal). Unlike classical micro-pipelining schemes, no logic redundancy needs to be introduced in order to have a high throughput execution in the fine-grained pipelined stages. In this paper, we discuss a routing architecture that supports the wave-steered LB architecture that was presented in [19].

We organize the rest of the paper as follows: Section 2 discusses previous work on throughput-intensive FPGAs. Section 3 describes the key architectural ideas on which our architecture is based. Section 4 summarizes the LB architecture [19] and describes the pipelined routing/interconnect fabric. Section 5 presents a characterization of applications that fit naturally in our architecture. Section 6 provides experimental results. This is followed by conclusions.

2 Previous Work

Previous work in the area of high throughput FPGAs has focussed mainly on the application end rather than on architecture. Retiming is a major technique used to achieve high throughputs in FPGAs. Von Herzen in [23] demonstrated a real-time DSP application (a 2-bit correlator) that could operate at 250 MHz on a 0.7µm Xilinx XC3000 device. Most of the work in [23] concentrated on choosing a cycle time and aggressively (manually) retiming the fairly systolic design to operate at this fast cycle. While past literature has dealt with pipelined memory lookup [10], no literature exists that studies pipelining inside the logic block of an FPGA.

Borriello et al in [5] presented a new FPGA architecture, Triptych, that blends logic and routing resources to achieve efficient implementation of a wide range of circuits in both area and speed. Their approach integrated mapping, placement and routing on this architecture to utilize the FPGA resources better. Our architecture, while fundamentally different in the internal operation of an individual Logic Block, has the same flavor in that it utilizes unused Logic Blocks for routing purposes. Another fundamental difference in our architecture is the use of pipelined interconnects.

Recent work has also dealt with the issue of throughput-intensive operation in the context of delays introduced by long interconnects. In [22], Tsu et al tackle the problem of pipelined interconnect in an effort to achieve high throughputs. Timing domains are determined by calculating the length of the interconnect that can be tolerated within a predetermined clock cycle. In this work, the authors design their Logic Blocks in a way that makes retiming possible. This retiming is necessary to accommodate the effect of a pipelined interconnect. While their Logic Block is essentially the same as the one found in present day Xilinx devices, they have a bank of flip-flops on their inputs instead of having a single optional output register on logic blocks. This gives the retiming needed to balance out path delay differences introduced by the

FPGA 2001, February 11-13, 2001, Monterey, CA, USA.

pipelined interconnect and allows throughputs of 250 MHz in 0.4µm technology. The results indicate an increase in the overall area and power due to the drastic increase in the number of flip-flops. The fact that this reconfigurable processor does not specifically target any single design domain, is also an inherent shortcoming of this architecture as far as performance is concerned. Our work, while similar in flavor to [22], targets a special class of designs. Section 5 explains our approach.

3 Wave-Steering

In [19], we proposed a high-throughput FPGA architecture in which LBs are complete binary trees wave-steered to the granularity of one level. Figure 1 shows a decision tree of three variables with each tree node directly mapped into Pass Transistor Logic (PTL), by replacing it with a 2:1 multiplexer followed by an inverter. This inverter accounts for voltage degradation between the levels. The tree is evaluated level by level, starting from the leaf nodes. Each level in the tree corresponds to a particular variable in the function. Logic '1' and '0' form the leaf nodes and the output of the node at the topmost level evaluates the function. In PTL mapped decision trees implementations of logic functions, the tree structures have input points physically distributed along the levels of the tree. Since the evaluation of the trees occurs in a bottom-up fashion, skewing of inputs fits naturally the evaluation, with each higher-level input skewed in to the tree after the previous-level input has already been fed.

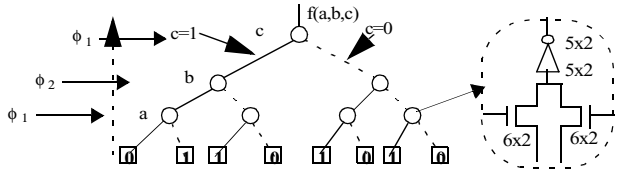


Figure 1: f(a,b,c) evaluated from bottom to top as a Decision Diagram Tree

In a conventional circuit, current data must propagate to the output latch of the circuit before the next wave of inputs can be pushed in. It is necessary to wait this long because for different input vectors, different input-output paths are activated and each path can have different delay. In this case, the throughput of the circuit equals its latency. However, if we can synthesize a fairly regular circuit such that all paths have almost equal delays, then more than one data wave can exist between two clock cycles. This is true because there is no need for the previous data to be latched into the output flip-flops before pushing in the next set of skewed inputs (in other words, internal node capacitances act as latches for the incoming waves). Although this may resemble a conventional pipelining scheme, it is fundamentally different in the sense that the input application points spatially follow the pipelined stages. Fine granularity pipelining of PTL mapped decision diagram structures inherently have input application points physically distributed along the stages of the pipeline, where each stage corresponds to a level characterized by a single variable. This variant of pipelining is called wave-steering[15][16].

The timing skew between two variables characterizing two successive stages (levels) in a wave-steered structure would typically be one stage delay. The skewing is accomplished by a chain of flip-flops and a unique clocking scheme. This will guarantee the operation of the circuit at a given frequency by construction. In a fully Wave-Steered Decision Tree, alternate Mux levels are called ϕ_1 and ϕ_2 levels. In any ϕ_1 level, the controlling input (or its complement) is logic '1' and data is propagated to the next level during this phase ϕ_1 of the clock. During phase ϕ_2 this particular level holds the logic level in the nfet output and inverter gate capacitances, as both the mux's selector lines remain at logic '0'. This provides the electrical isolations between successive data waves. Once the function has been partially evaluated for a particular variable in a level, only the combined information needs to be propagated and the value of that particular variable is no longer needed. The current input variable selects a path only after the previous variable in the vector had selected the correct path.

While wave-steering works best for designs without feedback, recently there have been two published works [24][25] that show that finite state machines (FSMs) can also be wave-steered. Both these works exhibited an increase in throughput performance by a factor of 3 for a similar increase in area. In this paper however, we restrict ourselves to designs without feedbacks.

4 FPGA Architecture

For sake of completeness, we briefly discuss the logic block architecture of our FPGA fabric [19]. For more detailed discussion on the internal operation of a LB, we refer the reader to [19]. Figure 2 shows a block diagram of a reconfigurable Logic Block (LB) slice.

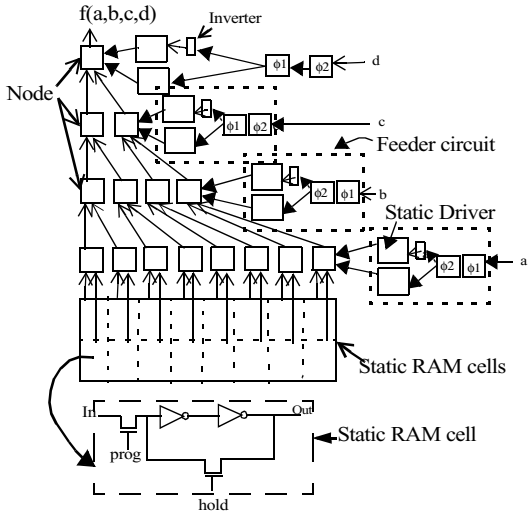


Figure 2: Block-Diagram of a Logic Block Slice

A Logic Block (LB) slice is a four-input LUT 1-output structure that can implement any function of upto four variables. Each of our logic blocks is composed of two such slices, with each slice built of a pass transistor logic (PTL) mapped 2:1 multiplexer binary decision tree. Each PTL mapped tree has four levels, with each level corresponding to an input variable. Alternate set of 2-levels are clocked by different phases (ϕ_1 and ϕ_2)of the clock. This means that unlike in [19], we wave-steer not every level but every 2 levels in the PTL tree. The usefulness of this approach will be evident in the next paragraph. Figure 2 also shows additional circuitry to skew signals that arrive earlier than they can be applied. This part of the LB slice is called the feeder circuit. There are two types of dynamic flip-flop cells in the feeder circuit, the F1 and F2. The F1-F2 pair will get the data out by the end of ϕ_2 phase while the F2-F1 phase will get the data out by the end of ϕ_1 phase. Each set of flip-flop cells feeds a clocked Nand gate and an inverter that act as a buffer assembly. This clocked Nand-inverter couple is used for driving the mux tree with the signal values during only one phase of the clock and blocking the data transfer during the alternate phase.

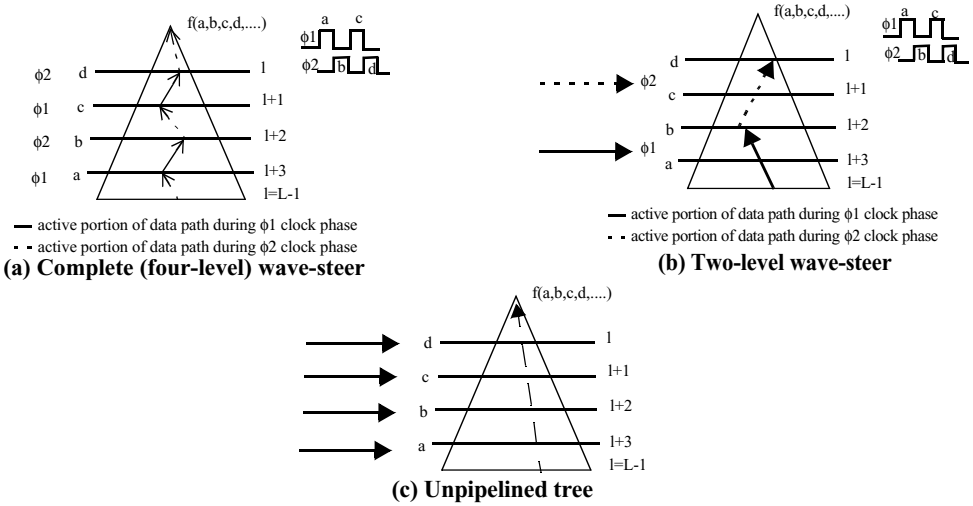


Figure 3: Wave-Steering Granularity

To determine the best throughput-latency trade-off, we performed a series of experiments to determine the level of wave-steering inside each logic block. A 4x4 array multiplier and its interconnect was used as the test-case for these sets of experiments. We manually mapped this design and varied the level of wave-steering inside each LB as shown in Figures 3.a-3.c. The resulting netlist in each case was simulated in HSPICE. All HSPICE simulations were done using a 0.25µm 5-metal CMOS technology at room temperature ($25°C$).Table 1 shows the result of this set of experiments. In all simulations, we have a setup and hold time of 0.1ns for the clock and input signals and this is maintained for all the remaining simulations in this paper.

Table 1: Steering Granularity

Pipeline Level	Cycle Time[a] (ns)	Power consumed (mW)
Four-level pipeline	1.1ns	4.13mW/cell @1.1ns cycle time
Two-level pipeline	1.2ns	2.05mW/cell @1.2ns cycle time
No Pipeline (LUT as a tree)	2.0ns	1.75mW/cell @2.0ns cycle time

a. This cycle time considers both the LBs and the neighboring interconnect.

A fully (four-level) wave-steered tree is shown in Figure 3.a. A two-level tree wave-steers every two levels of the decision tree (Figure 3.b) instead of every level while an unpipelined tree (Figure 3.c) has no wave-steering in any of the levels.

Using the unpipelined tree as the basis for the mapped design, the array multiplier can operate at a throughput of 2.0ns(500 MHz) while consuming 1.75mW/cell power as simulated in HSPICE. This power consumption figure is the average of all the LBs in our test-case operating at a cycle time of 2.0ns. A two-level wave-steer based setup gives a throughput of 1.2ns (833 MHz) and average power consumption of 2.05mW/cell at 833MHz, whereas a fully four-level wave-steered design gives a slightly higher throughput (910 MHz) while consuming 4.13mW/cell power at 910 MHz.

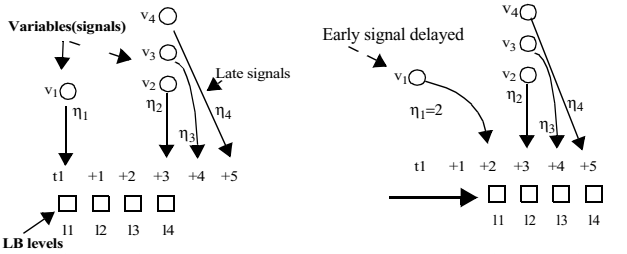
While it may appear counter-intuitive that a fully four-level pipelined tree has no inherent gain over a two-level pipeline scheme as far as throughput is concerned, the above result can be explained. In order to achieve the wave-steering effect in each tree, it is necessary to skew the input vectors in time using the feeder circuit. This feeder circuit drives the select lines of all the PTL multiplexers in each level in the tree. We performed HSPICE simulations to choose the minimum size transistors in the feeder circuit that can drive these select lines. In a four-level pipeline scheme, the time-skew between two consecutive levels is the time required by a data wave to travel from the bottom level to the upper level. In order for this skew to be minimum, the delay should be less than or equal to the delay through the feeder circuit. This means that when the data wave has propagated from the lower level to the higher level, we have to have a valid select signal at this higher level before the data wave can propagate further up the tree. If the delay through the feeder circuit is greater than the delay through each level in the tree, we have to increase the time-skew between the two consecutive levels to match the feeder-circuit delay in order to propagate the wave further up the tree. Hence there are two factors that determine the time skew -- the delay through each level of the tree and the delay through the feeder circuit. In the two-level wave-steer scheme, we are only wave-steering every second level in the tree. Thus we only need to skew in inputs corresponding to every second level in the tree. In this case, the delay through the feeder circuit is less than the delay through two levels of the tree. The time skew in this case is the time that the data wave needs to travel between every two-levels and is not dominated by the delay through the feeder circuit.

The potential advantage of choosing a two-level wave-steering scheme is evident from Table 1. When we steer only two levels in the tree, we use fewer flip-flops (this results in smaller LB area) as compared to the four-level Wave-Steered scheme while achieving nearly the same throughput. This translates into 50% less power consumption per cell on average. When compared to a fully unpipelined tree, we note that the two-level pipelined tree is 66.67% faster than the unpipelined tree at the cost of minimal increase in power consumption per cell. Even if we reason that in real life, 833 MHz in 0.25µm is optimistic since we have not considered some practical clocking and control signal issues, we can

still claim that a single 2-level Wave-Steered LB is 66.67% faster than a traditional unpipelined LB.

4.1 Wave-Steering without Stalling

A circuit is guaranteed to achieve a given clock period in our FPGA if the condition of wave-steering without stalling is met. This condition states that if all inputs to any LB have arrival times that are at most $\phi/2$ time units apart (ϕ is the clock period), then we can schedule these inputs in consecutive levels in the LBs. However, if some LB inputs do not meet this non-stalling condition, we can no longer guarantee the achievable clock period without introducing forced stalls in this circuit. These forced stalls are accomplished by pipeline switchpoints in our FPGA. Figure 4 shows an ordered list of input signals to an LB. The arriving signals introduce a stall in the LB. This means that there is at least one signal that arrives more than $\phi/2$ after the previous signal thereby increasing the LB latency. We move the early arriving signals forward (in Figure 4, signal 1) in time by the appropriate η_i half cycles (Figure 4.b) such that all arriving signals are in consecutive levels in the LB. We denote the modified arrival time of each signal as $t_i + \eta_i$ where t_i is the actual arrival time of signal i. All signal stalling is accomplished by pipelined switch points spread over the interconnect and discussed in the next sub-section.



(a) Actual Arrival times **(b) Modified Arrival times**

Figure 4: Actual and Modified Arrival times

4.2 Routing Architecture

Most commercial FPGA architectures are not well suited for throughput-intensive applications. In almost all instances, this can be attributed to the fact that their routing architectures are so general purpose that speed is often compromised. As pointed out in [22], there is really no reason while programmable logic should be an order of magnitude slower than its custom logic counterpart. While DSM technology can contribute to an increase in speed, it cannot get rid of long routing wires and switch boxes that run the length and breadth of a typical FPGA, thereby slowing it down. Keeping these factors in mind, we propose a routing architecture shown in Figure 5. We point out that our routing architecture is motivated by a category of high-speed designs that we target. As section 5 will show, these designs can be loosely categorized by a quantifying metric.

The routing architecture has 2 levels of hierarchy, with a single LB forming the basic element of the lower level hierarchy and a logic cluster forming the basic element of the higher level. Connections from the lower level hierarchy to the higher level hierarchy are accomplished through programmable wire taps (not shown). Each logic cluster consists of an array of 4x4 LBs having dedicated interconnect since high-speed DSP designs usually have local communication and involve iterative computation. Clusters of size sixteen(4x4) were chosen by determining the length of interconnect that can enclose these clusters without inducing unacceptable

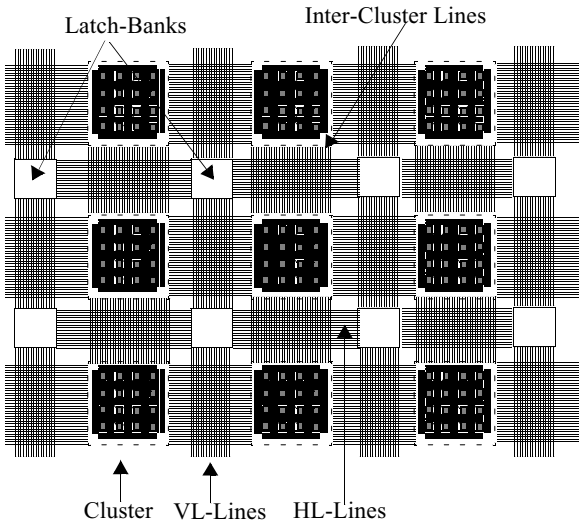
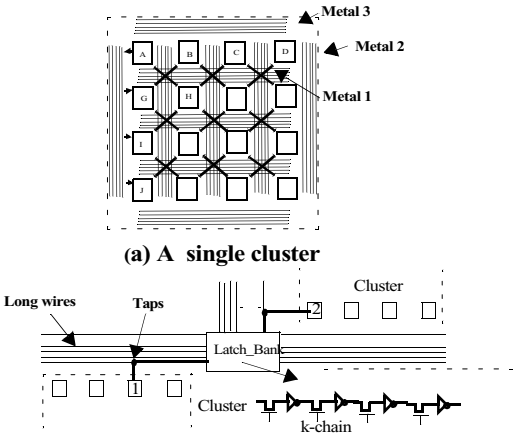


Figure 5: A hierarchical layout

delays. Inter-cluster routing is achieved using Vertical Long Lines (VL-Lines) in metal 1 and Horizontal Long Lines (HL-Lines) in metal 2. Power, ground and clock lines (not shown) are in metals 3, 4 and 5. Each segment of HL and VL lines is 0.5mm long and is pipelined using a bank of pipelined switchpoints called Latch-Banks. These latch-banks are pseudo-switch boxes having a programmable matrix such that track i in any HL/VL segment can only connect with track i in other wire segments surrounding the latch bank. Each switch point in the matrix is a pipeline point for track i and is k-deep, where k=4. This means, that any VL or HL line segment can use upto 4 clock period delays to stall any signal that is reaching its destination too early (see example in Figure 4), so that the wave-steering without stalling condition is satisfied. Each latch bank consists of a series of dynamic latches which can be tapped at different positions (upto k=4) through switches to get the desired delay. These dynamic latches are clocked by either the ϕ_1 or the ϕ_2 phase of the clock. Dedicated next level cluster routing is accomplished as shown in Figure 5. Each inter-cluster channel has a maximum channel width of 35 tracks while each intra-cluster in the lower hierarchical level has a channel width of 15 tracks. As section 6 will show, this is sufficient to route even random circuits. Routing inside a cluster is dedicated and restricted to next column/ row without any crossbar routing switches or interconnect pipelining.



(a) A single cluster

(b) Inter-cluster communication

Figure 6: Cluster communication

Figure 6.a shows how different LBs within a cluster are connected to each other. For example, LB_A can communicate with all LBs in its row/column as well as all LBs in the adjacent row/column. A particular logic block can place its output on the horizontal/vertical wires running along the cluster array and other neighboring blocks tap(through programmable switches) this signal. Figure 6.b shows an instance of inter-cluster communication. Programmable wire taps(switches) are used for tapping signals to/from long lines and signal delaying and bending is accomplished through taps(switches) in a k-deep (k=4) chain of dynamic latches. For simulation we model interconnects and wire taps as RC π chains (each interconnect has four such chains). In addition, each LB has a dedicated diagonal interconnect to its neighbor block to provide more versatility for routing designs which are systolic in nature.

Communication between any two LBs inside a cluster is full-duplex. In situations where dedicated routing resources are not enough or where the design under consideration doesn't map naturally into our fabric, we have the option of decreasing the logic density of a cluster and utilizing unused routing blocks as route-throughs in order to complete routing. Empty LBs have been used as route throughs in previous literature [5] and this concept is not unique to our architecture. Figure 7 shows routing through Logic Blocks inside a cluster. In this case LB 'C' is used as a route

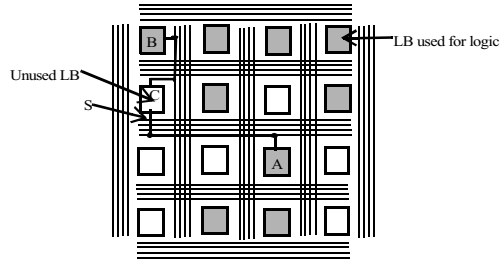


Figure 7: Unused LBs as route-throughs inside a cluster

through for signal S. 'A' places its output on the horizontal wire-mesh, 'C' taps it and routes it through as its output for B to tap it in again. Unused LBs can be used to bend signals in cases where a horizontal-vertical or vertical-horizontal communication is required. Since each LB has two slices, we can accomplish two signal bends per LB. Empty LBs can also be used for delaying certain signals that arrive too early in order to maintain the fine granularity pipelining without stalling inside each LB.

The motivation for using pipeline interconnects on the VL and HL lines comes from a desire to use these pipeline points for synchronization purposes in the case where signal sources are at different levels and have different delays. For example, a signal that is arriving earlier than other signals whose sinks are in the same cluster, will have to be delayed to avoid unnecessary stalls and increase in latency later on (see Figure 4). Since wave-steering imposes strict timing windows on the architecture, this pipelining is essential. This also avoids the possibility of data getting corrupted due to the presence of multiple data waves at the same time without isolation. This opens an interesting possibility of not only having a hierarchical structure, but also having hierarchical wave-steering, i.e. not only at the LB level (as is the case at present) but also at the cluster level. This is currently being investigated as are other issues related to clock and control signal distribution.

5 Design Characterization

To take advantage of the wave-steering design style which inherently favors high throughput applications, we need to find a quantifying metric that can categorize applications that would naturally fit in our FPGA architecture. Through experimentation, we find that our architecture is a good fit for designs in which connection between different blocks are constrained to be local (not necessarily next neighbor) and single design modules have similar complexity (note that this does not imply the same functionality on different blocks). We call these designs *regular or feasible designs*. This basically says that feasible designs have local connectivity between their modules and demonstrate an iterative nature that can usually be captured by a few regular patterns. This iterative nature and local inter-connectivity leads to circuits having almost equal delays along all paths. Such feasible designs exhibit certain characteristics that distinguish them from random designs. To quantify this difference between feasible and random designs, we use an empirical relationship known as Rent's rule [12] developed for predicting the number of external connections from a given number of components in a circuit. This relationship is:

$$I = bC^p \quad \textbf{(1)}$$

where

I is the number of external connections,
C is the average number of components in a circuit,
b is the average number of connections per component,
p is a small positive exponent, known as the Rent's exponent.

Rent's rule is also used to study the distribution of connection lengths in designs. We use this relationship to characterize feasible circuits. We experimented on two sets of designs: a) systolic and arithmetic designs and b) random designs from the MCNC 91 benchmarks to categorize designs according to their value of *p*. The feasible designs consisted of an array multiplier, a 2-bit correlator, a 3-tap systolic FIR filter, an image template matcher, a 1-D convolution and a vector quantizer. The underlying characteristics of all these designs is the iterative nature of their modules and short predictable interconnect between these modules. These designs were implemented in our architecture, with each design module mapping into several LB clusters. Table 2 shows the I/O characteristics of these manually mapped designs into clusters of sixteen LBs. The size of these designs ranges from 64 LBs (multi-

Table 2: Rent's exponent for regular designs

Circuit	I/O signals	Logic Blocks used/ cluster	*p*
4x4 array multiplier	12	16	0.41
2-bit correlator	9	11	0.34
FIR Filter	12	15	0.41
Template matcher	13	16	0.43
Convolution	13	16	0.43
Quantizer	12	15	0.41

plier) to 6336 LBs (template matcher). Since these designs are feasible designs, almost all LBs in a cluster are used for logic (except in the case of the correlator where 5 LBs are used for stalling the pipeline and bending signals). The third and fourth columns show the average I/O signals and LBs used per cluster, for all the designs. The average value of p for these feasible designs was found to be 0.41. This is in line with the notion that these designs usually have values of $p < 0.5$ since their modules are more likely to fan out to modules only within their local neighborhood. On the contrary, our experiments (Section 6) show that the benchmark random designs have on average a value of $p = 0.6$. Past literature also exists [12] which shows that random designs usually exhibit a value of 0.5<=p<=1.0. The locality of a connection is determined by how far a given module can fanout under the timing constraints dictated by the clock cycle time. This locality measure also reflects the fact that signals have to arrive at input points of LBs at the correct time in order to maintain proper functionality in our architecture. A range of systolic [13] and arithmetic circuits fall into the category of feasible designs that require high throughputs. These include most DSP applications that inherently and iteratively use multiply and accumulate units and are fairly regular. Applications such as FIR and IIR filters, data compressing, encrypting, image template matching and data searching designs would naturally fit in our architecture. These applications use only a few simple modules (mostly multiply and accumulate), can be expressed in a systolic fashion if necessary and can also be serialized in cases where high throughput can be traded for a similar decrease in area.

6 Results

To test the feasibility of our proposed architecture for throughput-intensive applications, we laid out our FPGA logic block and preliminary interconnect scheme in a 0.25µm CMOS 5-metal technology. Each LB occupies 720λ x 920λ. Assuming a die size of 1.5cm x 1.5cm, the chip can hold approximately 6400 such LBs (including estimated area for interconnects and programming wires) or

400 higher level clusters. We simulated the extracted SPICE netlist of a LB and its neighboring interconnect and used this netlist to validate four designs in our Wave-Steered FPGA fabric (we used the routing architecture shown in Figure 5). These include a 4x4 array multiplier, a 2-bit correlator, a 3-tap bit level systolic FIR filter and an 8x8 image template matcher. Note that all these designs conform to the feasibility analysis in Section 5 and fit naturally in our architecture. With budgeting for programming wires and delay induced by clock skew, all these designs can achieve throughputs of 625 MHz in 0.25µm CMOS 5-metal technology at room temperature. To get a picture of how performance is affected with rise of on-chip temperature, we performed SPICE simulations at temperatures from 25 to $125°C$. Figure 8 shows how clock period and power consumption per LB vary as temperature is increased. These simulations were performed on a 4x4 pipelined array multiplier without considering any clock skews. The first graph shows a performance degradation of 45% at $125°C$ over the performance at $25°C$ in 0.25µm technology. Other processes (0.35µm and 0.5µm) show a similar trend. The second graph shows how power consumption per LB varies as temperature is increased. Here, power consumption at $125°C$ decreases by 32% over power consumption at $25°C$ in 0.25µm technology when operating at 625 MHz. Note that all 3 technologies have different supply voltages and hence the noticeable differences in dynamic power dissipations.

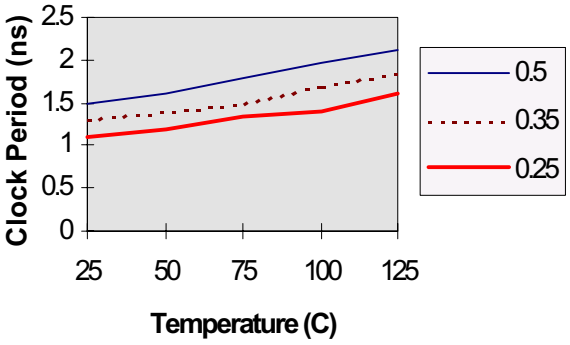


Figure 8.a: Performance vs. Temperature

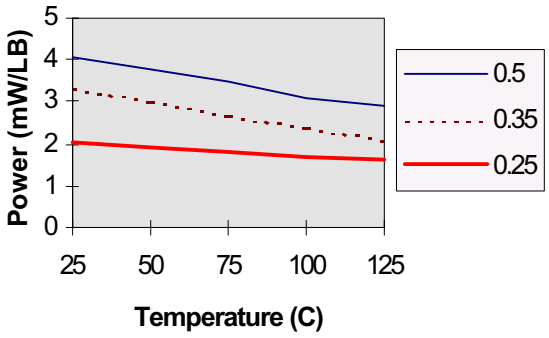


Figure 8.b: Power vs. Temperature

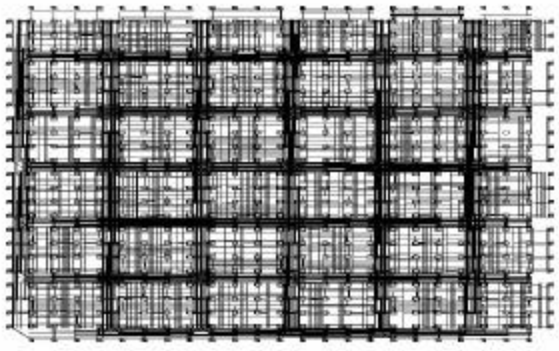


Figure 9: A random circuit placed and routed in clusters of 4x4

In order to exhibit the ability of this architecture to handle random designs (p > 0.5), we placed and routed some MCNC benchmarks on our fabric. Table 3 shows the results. Figure 9 shows one such circuit placed and routed in this architecture. These results are obtained when we restrict the number of unique inter-cluster signals to 24. Increasing this limit renders some benchmark circuits unrouteable. We see an average logic utilization of 55.3% (8.8 LBs/cluster) per cluster leaving 7.2 LBs/cluster for routing. In all of the circuits in Table 3, almost all LBs not configured for logic purposes are used for routing inside the cluster. To guarantee the operation at a given frequency for these random circuits, we show that the given placed and routed circuit can operate without any

stalls at the LBs. This means that the pipeline elements should account for delaying early arriving signals in such a way that all signals at all LB input points arrive consecutively (i.e. within $\phi/2$). Table 3 shows the number of such interconnect latches needed to ensure the wave-steering without stalling condition in random circuits. Note that we do not discuss interconnect latch counts for feasible circuits because in our tested feasible circuits, signals inherently tend to arrive at the required times and almost all signal skewing/delaying is accomplished using the latches already present in the LB feeder circuits. For random circuits, latch counts are found from the arrival times at all LB inputs and then dividing these arrival times into time domains dictated by the specified clock period. The MCNC benchmarks can operate at a clock period of 250 MHz. This clock period was arrived at by examining the trade-offs between a faster clock speed and using more interconnect latches(Figure 10).

Table 3: Random Circuits

Circuit	PI/PO	#LBs	Max 24 signals				
			# Clusters	LBs/cluster	Inter-cluster Channel width	Avg. #I/#O	Latches
C499	41/32	74	12	6.2	27	11.3/5.1	50
C432	36/7	119	13	9.2	26	11.9/6.6	142
C6288	32/32	528	49	10.7	30	11.6/6.3	1364
C7552	207/108	881	90	9.7	34	11.7/7.4	236
alu4	18/8	584	78	7.5	28	9.7/6.2	496
alu2	10/6	355	38	9.3	23	11.1/7.8	214
C1908	33/25	224	24	9.3	31	11.1/7.5	166
C3540	50/22	509	61	8.3	33	11.4/7.1	476
Avg.	**54/30**	**409.25**	**45.63**	**8.8**	**27.75**	**11.2/6.7**	**393**

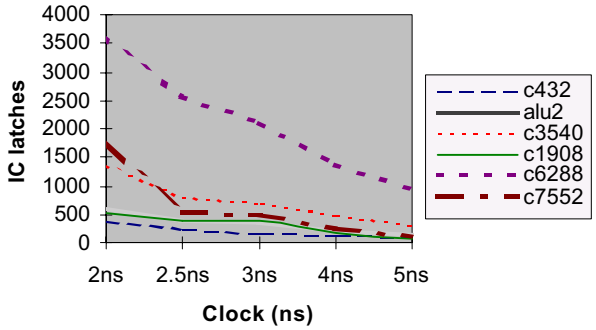


Figure 10: Latch-Frequency tradeoff

There are 35 tracks per inter-cluster channel and each latch bank is 4-deep, hence, there is no latch congestion in any of these latch-banks when the clock period is fixed at 250 MHz. Since the quality of the placement solution directly affects the number of interconnect latches that are used, we are currently modifying our place and route tool based on [3] to improve our solution quality.

7 Conclusions

In this paper, we discuss a pipelined interconnect architecture to support a wave-steered FPGA fabric [19]. We categorize designs into feasible and random designs based on the Rent's parameter and show that both these designs can be routed in our FPGA fabric under the wave-steering without stalling condition, with the random designs operating at speeds of 250MHz and the feasible designs operating at 625 MHz in 0.25µm technology. This noticeable difference in operating speeds highlights the attractiveness of our architecture for feasible designs that can benefit most in a throughput-intensive environment. Further, we show that as on-chip temperatures increase, there is as much as 45% performance degradation in performance over the room-temperature performance. Future work in this area will focus on a few key architectural improvements, including using LBs without feeder circuit latches and reordering of LB input signals for latch minimization. Clock and control signal distribution as well as placement issues are also under investigation.

7 Acknowledgment

This work was supported in part by the NSF grant CCR 9811528 and in part by California MICRO program through Xilinx.

8 References

[1] V. Bertacco, "Decision Diagrams and Pass Transistor Logic Synthesis", Proc. of the ACM/IEEE International Workshop on Logic Synthesis, May 1997, pp. 1-5.

[2] V. Betz, J. Rose, A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publishers, 1999.

[3] V. Betz, J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research", 7th International Workshop FPLA, 1997, pp. 213-222.

[4] E. I. Boemo, S. Lopez-Buedo, J. M. Meneses, "Some Experiments About Wave-Pipelining FPGA's", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.6, No.2, June 1998, pp. 232-237.

[5] G. Borriello, C. Ebeling, S.A. Hauck, S. Burns, "The Triptych FPGA Architecture", IEEE Transactions on VLSI Systems, Vol.3, No.4, Dec. 1995, pp.491-501.

[6] R.E. Bryant, "Graph-based Algorithms for Boolean functions manipulation", IEEE Transactions on Computers, Vol. C-35, Aug. 1986, pp. 677-691.

[7] P. Buch et al, "On Synthesizing Pass Transistor Networks", Proc. of the ACM/IEEE International Workshop on Logic Synthesis, May 1997, pp. 1-8.

[8] P. Buch, A. Narayan, A.R. Newton, A. Sangiovanni-Vincentelli, "Logic Synthesis for Large Pass Transistor Circuits", Proc. IEEE International Conference on Computer Aided-Design (ICCAD), November 1997, pp. 663-670.

[9] W.P. Burleson, M. Ciesielski, F. Klass, W. Liu, "Wave-Pipelining:A Tutorial and Research Survey", IEEE Transactions on VLSI Systems, Vol.6, No.3, Sep.'98, pp. 464-470.

[10] T. Chappell, "A 2-ns cycle, 3.8ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture", IEEE Journal of Solid-State Circuits, vol 26 (no.11), Nov 1991, pp.1577-85.

[11] J. Cong, Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", IEEE Transactions on Computer-Aided Design (CAD), 13(1):1-12, January 1994, pp. 1-12.

[12] M. Feuer, "Connectivity of Random Logic", IEEE Transactions on Computers, vol. C-J31, no.1, January 1982, pp.29-33.

[13] S.Y. Kung, "VLSI Array Processors", Prentice Hall, 1988.

[14] W.K.C. Lam, R.K.Brayton and A.L.Sangiovanni-Vincentelli, "Valid Clock Frequencies and Their Computation in Wave

Pipelined Circuits", IEEE Transactions on CAD of IC and Systems, Vol. 15, No.7, July 1996, pp. 791-807.
[15] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, S.I. Long, "Wave Steering in YADDs: A Novel Non-iterative Synthesis and Layout Technique", Proc. IEEE Design Automation Conference, June 1999, pp 466-471.
[16] A. Mukherjee, M. Marek-Sadowska, S.I. Long,"Wave Pipelining YADDs- A Feasibility Study", Proc. IEEE Custom Integrated Circuits Conference, May 1999, pp 559-562.
[17] U. Peisl, "Image Correlation using a bit level systolic array", IEEE International Symposium on Circuits and Systems, June 1988, pp. 2689-2693.
[18] R. Ray, "A Placement and Routing Algorithm for a new High Throughput FPGA", MS Thesis, UCSB 1999.
[19] A. Singh, L. Macchiarulo, A. Mukherjee, M. Marek-Sadowska, "A Novel High Throughput Reconfigurable FPGA Architecture", Eighth ACM International Symposium on FPGAs, February 2000, pp.22-27.
[20] R. Sudhakar, "YADDA: Layout Synthesis using Pass Transistor Logic", MS Thesis, UCSB, 1998.
[21] K. Taki, "A Survey for Pass-Transistor Logic Technologies", Proc. Asia South-Pacific Design Automation Conference, February 1998, pp. 223-226.
[22] W. Tsu, K. Macy, A. Joshi, R. Huang, A. DeHon, "HSRA: High Speed, Hierarchical Synchronous Reconfigurable Array", Seventh ACM International Symposium on FPGAs, February 1999, pp. 125-134.
[23] B.Von Herzen, "Signal Processing at 250 MHz Using High-Performance FPGAs", IEEE Transactions on VLSI Systems, Vol. 6. No.2, June '98, pp. 238-246.
[24] L. Macchiarulo, S.M. Shu, M. Marek-Sadowska, "Wave-Steered FSMs", proc. DATE 2000, pp.270-276.
[25] L. Macchiarulo, M. Marek-Sadowska, "Wave-Steering One-hot Encoded FSMs", proc. 37th Design Automation Conference, June 2000, pp. 357-360.
[25] The Programmable Logic Data Book, Xilinx Inc. 1999.
[26] URL: http://www.xilinx.com/products/logicore/lcoredes.htm